SYSTEM AND METHOD FOR WEB SERVING

BACKGROUND OF THE INVENTION

1. Related Applications.

The present invention claims priority from U.S. Provisional Patent Application No. 60/197,490 entitled CONDUCTOR GATEWAY filed on April 17, 2000.

2. Field of the Invention.

The present invention relates, in general, to network information access and, more particularly, to software, systems and methods for serving web-based content from a plurality of front-end computers sharing a database.

3. Relevant Background.

Increasingly, business data processing systems, entertainment systems, and personal communications systems are implemented by computers across networks that are interconnected by internetworks (e.g., the Internet). The Internet is rapidly emerging as the preferred system for distributing and exchanging data. Data exchanges support applications including electronic commerce, broadcast and multicast messaging, videoconferencing, gaming, and the like. In electronic commerce (e-commerce) applications, it is important to provide a satisfying buying experience that leads to a purchase transaction. To provide this high level of service, a web site operator must ensure that data is delivered to the customer in the most usable and efficient fashion.

The Internet is a collection of disparate computers and networks coupled together by a web of interconnections using standardized communications protocols. While most Internet access is currently performed using conventional personal computers and workstations, the variety of devices that access the Internet is growing quickly and expected to continue to grow. Wireless devices such as telephones and pagers are increasingly common. It is expected that a variety of appliances and devices within offices, businesses, and households will support Internet connectivity in the coming years. As a result, supporting interfaces to these devices has become a significant issue in the problems associated with providing services over the Internet, including database services. For example, it is desirable to format data returned to a cellular phone or pager much differently than might be desirable on a workstation having a high bandwidth connection and advanced graphics processing capability. Conventional database systems do not recognize the different demands required by such diverse access devices.

The Internet is characterized by its vast reach as a result of its wide and increasing availability and easy access protocols. Unfortunately, the ubiquitous nature of the Internet results in variable bandwidth and quality of service between points. The latency and reliability of data transport is largely determined by the total amount of traffic on the Internet and so varies wildly seasonally and throughout the day. Other factors that affect quality of service include equipment outages and line degradation that force packets to be rerouted, damaged and/or dropped. Also, routing software and hardware limitations within the Internet infrastructure may create bandwidth bottlenecks even when the mechanisms are operating within specifications. The variable nature of the quality of

service (QOS) provided by the Internet has made development and deployment of database system that leverage the Internet infrastructure difficult.

Currently, World Wide Web services are implemented as client-server systems. The client is typically implemented as a web browser application executing on a network-connected workstation or personal computer. The server is typically implemented as a web server at a fixed network address. A client enters a uniform resource locator (URL) or selects a link pointing to a URL where the URL identifies the web server and particular content from the server that is desired. The client request traverses the network to be received by the server.

The web server then obtains data necessary to compose a web page responsive to the client request. In the case of static pages, the web server may simply retrieve the page from a file system, and send it in an HTTP response packet using conventional TCP/IP protocols and interfaces. In the case of dynamically generated pages, the web server obtains data necessary to generate a responsive page, typically through one or more database accesses. The web server then generates a page, typically a markup language document, that incorporates the retrieved data. Once generated, the web server sends the dynamic page in a manner similar to a static page.

One problem with the existing system is that the web server activities required to generate a page can be time consuming. Web servers can become overburdened and fail when their limited connection and processing resources are exceeded. To ensure performance, web site operators often build in excess capacity increasing the cost and complexity of implementing a web site.

Another limitation lies in the fact that the web server is typically configured to generate only one type of page for a given request. Generally, it is left to the browser software to display the generated page in an expected and useful manner. This, however, may not be possible in many cases due to the wide variety of display devices currently used to execute web browser clients. For example, a personal digital assistant (PDA), telephone, or pager cannot be expected to display an HTML page in a similar manner to a workstation or personal computer. Even on a personal computer, varieties of browser software often cause conflicting goals in page generation within the web server.

Some web servers can be programmed to respond to parameters sent with a client request to generate a page more specifically tailored to the requester's needs. For example, an HTTP request parameter may specify the browser version and maker, or may specify that the browser is operating on a wireless device. Such implementations require the web server to be specially programmed for each device type. Maintaining such functionality and creating new functionality for new devices is a significant maintenance problem and, in practice, limits a web site's ability to reach new audiences.

The web server interface often becomes a critical bottleneck in database performance. Web servers have limited resources (e.g., buffers, ports, etc.) that must be shared amongst the tasks of maintaining connections, processing requests, accessing data, and rendering pages. These resources include software and hardware resources within the web server that create and maintain connections to clients, as well as resources used to translate requests into a database recognized format and translate

responses to formats that can be recognized by clients. Multiple web servers may be required to support even modest activity rates. This becomes difficult and expensive for the web site owner to establish and administer.

A particular need exists in environments that involve multiple users accessing a shared network resource such as a database server or database management system. Examples include broadcast, multicast and videoconferencing as well as most electronic commerce (e-commerce) applications. These environments increasingly involve resource-intensive processing at the web server.

SUMMARY OF THE INVENTION

Briefly stated, the present invention involves a method and system for serving web-based content over a communication network. Requests for web content are generated using a plurality of client applications coupled to the network. An intermediary server is coupled to the network to receive the requests for web content from client applications. A data server is coupled to the network and having an interface for communicating with the intermediary server. The intermediary server accesses the data server in response to receiving a request from a client application. Using the intermediary server, a web page is generated using the database content obtained from the data server. The web page is delivered to the client application that generated the request for database content.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a general distributed computing environment in which the present invention is implemented;

FIG. 2A shows in block-diagram form significant components of a system in accordance with a first embodiment of the present invention;

FIG. 2B shows in block-diagram form an arrangement of components in accordance with a second embodiment of the present invention;

FIG. 3 shows a domain name system used in an implementation of the present invention;

FIG. 4A shows intermediary server components of FIG. 2A in greater detail;

FIG. 4B shows front-end components of FIG. 2B in greater detail; and

FIG. 5 shows back-end components of FIG. 2B in greater detail.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention is illustrated and described in terms of a distributed computing environment such as an enterprise computing system using public communication channels such as the Internet. However, an important feature of the present invention is that it is readily scaled upwardly and downwardly to meet the needs of a particular application. Accordingly, unless specified to the contrary, the present invention is applicable to significantly larger, more complex network environments,

including wireless network environments, as well as small network environments such as conventional LAN systems.

A web server or web site can be thought of conceptually as a set of data and a bundle of services that manipulate that data to generate web pages. Traditionally, the data and program code that implement the services are housed together on a server. In early web site implementations this co-location of services and data was viewed as practical.

The present invention recognizes that the services used to manipulate the data can be more efficiently implemented in distributed servers located throughout a network topology. These distributed servers access one or more shared databases to generate web pages in response to requests. Because the computers implementing the services are separate from the computers implementing the database, each can be optimized for the specialized task required. Resources used to generate pages will not conflict with resources used to perform database queries, for example.

Moreover, the "front-end" computers and intermediate computers used to generate pages can be made more specific to a set of client applications. For example, a first front-end computer might serve cell phone clients, while a second front-end server might serve workstation clients. The first front-end server can implement special-purpose functionality to improve display and performance for the wireless clients without special-purpose programming on the part of the web-site owner.

FIG. 1 shows an exemplary computing environment 100 in which the present invention may be implemented. Environment 100 includes a plurality of local networks such as Ethernet network 102, FDDI network 103 and Token

ring network 104. Essentially, a number of computing devices and groups of devices are interconnected through a network 101. For example, local networks 102, 103 and 104 are each coupled to network 101 through routers 109. LANs 102, 103 and 104 may be implemented using any available topology and may implement one or more server technologies including, for example UNIX, Novell, or Windows NT networks, or peer-to-peer type network. Each network will include distributed storage implemented in each device and typically includes some mass storage device coupled to or managed by a server computer. Network 101 comprises, for example, a public network such as the Internet or another network mechanism such as a fibre channel fabric or conventional WAN technologies.

Local networks 102, 103 and 104 include one or more network appliances 107. One or more network appliances 107 may be configured as an application and/or file server. Each local network 102, 103 and 104 may include a number of shared devices (not shown) such as printers, file servers, mass storage and the like. Similarly, devices 111 may be shared through network 101 to provide application and file services, directory services, printing, storage, and the like. Routers 109 provide a physical connection between the various devices through network 101. Routers 109 may implement desired access and security protocols to manage access through network 101.

Network appliances 107 may also couple to network 101 through public switched telephone network 108 using copper or wireless connection technology. In a typical environment, an Internet service provider 106 supports a connection to network 101 as well as PSTN 108 connections to network appliances 107.

Network appliances 107 may be implemented as any kind of network appliance having sufficient computational function to execute software needed to establish and use a connection to network 101. Network appliances 107 may comprise workstation and personal computer hardware executing commercial operating systems such as Unix variants, Micrsosoft Windows, MacIntosh OS, and the like. At the same time, some appliances 107 comprise portable or handheld devices using wireless connections through a wireless access provider such as personal digital assistants and cell phones executing operating system software such as PalmOS, WindowsCE, EPOC and the like. Moreover, the present invention is readily extended to network devices such as office equipment, vehicles, and personal communicators that make occasional connection through network 101.

Each of the devices shown in FIG. 1 may include memory, mass storage, and a degree of data processing capability sufficient to manage their connection to network 101. The computer program devices in accordance with the present invention are implemented in the memory of the various devices shown in FIG. 1 and enabled by the data processing capability of the devices shown in FIG. 1. In addition to local memory and storage associated with each device, it is often desirable to provide one or more locations of shared storage such as disk farm (not shown) that provides mass storage capacity beyond what an individual device can efficiently use and manage. Selected components of the present invention may be stored in or implemented in shared mass storage.

One feature of the present invention is that front-end servers 201 (shown in Fig. 2B) and/or intermediate servers 206 (shown in Fig. 2A) are implemented as an

interchangeable pool of servers, any one of which may be dynamically configured to provide the database application services. The embodiments of FIG. 2A and FIG. 2B are not strictly alternative as they may coexist in a network environment. A redirection mechanism is enabled to select from an available pool of front-end servers and direct client request packets from the originating web server to a selected front-end server 201 or intermediary server 206.

In the case of web-based environments, front-end 201, intermediary server 206, and back-end server 203 are implemented using custom or off-the-shelf web server software. For purposes of this document, a web server is a computer or system of computers running server software coupled to the World Wide Web (i.e., "the web") that delivers or serves web pages. The web server has a unique IP address and accepts connections in order to service requests by sending back responses. A web server differs from a proxy server or a gateway server in that a web server has resident a set of resources (i.e., software programs, data storage capacity, and/or hardware) that enable it to execute programs to provide an extensible range of functionality such as generating web pages, accessing remote network resources, analyzing contents of packets, reformatting request/response traffic and the like using the resident resources. In contrast, a proxy simply forwards request/response traffic on behalf of a client to resources that reside elsewhere, or obtains resources from a local cache if implemented. A web server in accordance with the present invention may reference external resources of the same or different type as the services requested by a user, and reformat and augment what is provided by the external resources in its response to the user. Commercially available web server software

includes Microsoft Internet Information Server (IIS), Netscape Netsite, Apache, among others. Alternatively, a web site may be implemented with custom or semi-custom software that supports HTTP traffic.

In the embodiment of Fig. 2A, intermediary servers 206 interact directly with data servers 210, 211 and 212. In the embodiment of Fig. 2B, the front-end server establishes and maintains an enhanced communication channel with a back-end server 203. By enhanced, it is meant that the channel offers improved quality of service, lower latency, prioritization services, higher security transport, or other features and services that improve upon the basic transport mechanisms (such as TCP) defined for Internet data transport.

In the specific examples herein, client 205 comprises a network-enabled graphical user interface such as a World Wide Web browser ("web browser"). However, the present invention is readily extended to client software other than conventional web browser software. Any client application that can access a standard or proprietary user level protocol for network access is a suitable equivalent. Examples include client applications that act as front ends for file transfer protocol (FTP) services, voice over Internet protocol (VoIP) services, network news protocol (NNTP) services, multi-purpose internet mail extensions (MIME) services, post office protocol (POP) services, simple mail transfer protocol (SMTP) services, as well as Telnet services. In addition to network protocols, the client application may serve as a front-end for a network application such as a database management system (DBMS) in which case the client application generates query language (e.g., structured query language or "SQL") messages. In wireless appliances, a client

application functions as a front-end to a wireless application protocol (WAP) service.

Data storage mechanism 210, virtual database server 211, and database server 212, collectively referred to as "data servers", implement connectivity to network devices such as back-end 203 and/or intermediary server 206 to receive and process requests for data. Data servers can be implemented as a database including relational, flat, and object oriented databases. Alternatively, data servers may comprise a virtual database that accesses one or more other databases. Further, data servers may be a data storage device or network file system that responds to requests by fetching data.

Fig. 2B illustrates an embodiment in which intermediary server 206 is implemented by cooperative action of a front-end computer 201 and a back-end computer 203. Front-end mechanism 201 serves as an access point for client-side communications. In one example, front-end 201 comprises a computer that sits "close" to clients 205. By "close", "topologically close" and "logically close", it is meant that the average latency associated with a connection between a client 205 and a front-end 201 is less than the average latency associated with a connection between a client 205 and data servers 210-212. Desirably, front-end computers have as fast a connection as possible to the clients 205. For example, the fastest available connection may be implemented in the point of presence (POP) of an Internet service provider (ISP) 106 used by a particular client 205. However, the placement of the front-ends 201 can limit the number of browsers that can use them. Because of this, in some applications it may bemore practical to place one front-end computer in such a way that several POPs can connect to it. Greater distance

between front-end 201 and clients 205 may be desirable in some applications as this distance will allow for selection amongst a greater number front-ends 201 and thereby provide significantly different routes to a particular back-end 203. This may offer benefits when particular routes and/or front-ends become congested or otherwise unavailable.

Transport mechanism 202 is implemented by cooperative actions of the front-end 201 and back-end 203. Back-end 203 processes and directs data communication to and from data servers 210-212. Transport mechanism 202 communicates data packets using a proprietary protocol over the Internet infrastructure in the particular example. Hence, the present invention does not require heavy infrastructure investments and automatically benefits from improvements implemented in the general-purpose network 101. Unlike the general-purpose Internet, front-end 201 and back-end 203 are programmably assigned to serve accesses to a particular data server 210-212 at any given time.

It is contemplated that any number of front-end and back-end mechanisms may be implemented cooperatively to support the desired level of service required by the data server owner. The present invention implements a many-to-many mapping of front-ends to back-ends. Because the front-end to back-end mappings can by dynamically changed, a fixed hardware infrastructure can be logically reconfigured to map more or fewer front-ends to more or fewer back-ends and web sites or servers as needed.

Front-end 201 together with back-end 203 function to reduce traffic across the transport morphing protocol™ (TMP™) link 202 and to improve response time for selected browsers. Transport morphing protocol and TMP are

trademarks or registered trademarks of Circadence Corporation in the United States and other countries. Traffic across the TMP link 202 is reduced by compressing data and serving browser requests from cache for fast retrieval. Also, the blending of request datagrams results in fewer request:acknowledge pairs across the TMP link 202 as compared to the number required to send the packets individually between front-end 201 and back-end 203. This action reduces the overhead associated with transporting a given amount of data, although conventional request:acknowledge traffic is still performed on the links coupling the front-end 201 to client 205 and back-end 203 to a web server. Moreover, resend traffic is significantly reduced further reducing the traffic. Response time is further improved for select privileged users and for specially marked resources by determining the priority for each HTTP transmission.

In one embodiment, front-end 201 and back-end 203 are closely coupled to the Internet backbone. This means they have high bandwidth connections, can expect fewer hops, and have more predictable packet transit time than could be expected from a general-purpose connection. Although it is preferable to have low latency connections between front-ends 201 and back-ends 203, a particular strength of the present invention is its ability to deal with latency by enabling efficient transport and traffic prioritization. Hence, in other embodiments front-end 201 and/or back-end 203 may be located farther from the Internet backbone and closer to clients 205 and/or web servers 210. Such an implementation reduces the number of hops required to reach a front-end 201 while increasing the number of hops within the TMP link 202 thereby yielding control over more of the transport path to the management mechanisms of the present invention.

Clients 205 no longer conduct all data transactions directly with the data servers 210-212. Instead, clients 205 conduct some and preferably a majority of transactions with front-ends 201, which simulate, emulate, or actually implement the functions of data servers 210-212. Client data is then sent, using TMP link 202, to the back-end 203 and then to the data servers 210-212. Running multiple clients 205 over one large connection provides several advantages:

- Since all client data is mixed, each client can be assigned a priority. Higher priority clients, or clients requesting higher priority data, can be given preferential access to network resources so they receive access to the channel sooner while ensuring low-priority clients receive sufficient service to meet their needs.

- The large connection between a front-end 201 and back-end 203 can be permanently maintained, shortening the many TCP/IP connection sequences normally required for many clients connecting and disconnecting.

A particular advantage of the architectures shown in FIG. 2A and FIG. 2B is that they are readily scaled. In accordance with the present invention, not only can the data itself be distributed, but the functionality and behavior required to generate web pages from the data is readily and dynamically ported to any of a number of intermediary computers 206 and/or front-ends 201 and/or back-ends 203. In contrast, conventional web server systems lock the functionality to a particular server or limited set of servers. In this manner, any number of client machines 205 may be supported. In a similar manner, a database owner may choose use multiple data servers 210-212 that are co-located or distributed

throughout network 101. To avoid congestion, additional front-ends 201 may be implemented or assigned to particular databases. Each front-end 201 and/or intermediary server 206 is dynamically re-configurable by updating address parameters to serve particular web sites. Client traffic is dynamically directed to available front-ends 201 to provide load balancing and quality of service (QoS) management. Hence, when quality of service drops because of a large number of client accesses to a particular data server 210-212, an additional front-end 201 and/or intermediary server 206 can be assigned to the data server 210-212 and subsequent client requests directed to the newly assigned computer to distribute traffic across a broader base.

In the examples, dynamic configuration is implemented by a front-end manager component 207 (shown only in FIG. 2B) that communicates with multiple front-ends 201 to provide administrative and configuration information to front-ends 201. Each front-end 201 includes data structures for storing the configuration information, including information identifying the IP addresses of data servers 210-212 to which they are currently assigned. Other administrative and configuration information stored in front-end 201 and/or intermediary servers 206 may include information for prioritizing particular data from and to particular clients, quality of service information, and the like.

Similarly, additional back-ends 203 can be assigned to a database to handle increased traffic. Back-end manager component 209 couples to one or more back-ends 203 to provide centralized administration and configuration service. Back-ends 203 include data structures to hold current configuration state, quality of service

information and the like. In the particular examples front-end manager 207 and back-end manager 209 serve multiple data servers 210-212 and so are able to manipulate the number of front-ends and back-ends assigned to each data server 210-212 by updating this configuration information. When the congestion for the data server 210-212 subsides, the front-end 201, back-end 203, and/or intermediary server 206 can be reassigned to other, busier databases. These and similar modifications are equivalent to the specific examples illustrated herein.

In order for a client 205 to obtain service from a front-end 201, it must first be directed to a front-end 201 that can provide the desired service. Preferably, client 205 does not need to be aware of the location of front-end 201, and initiates all transactions as if it were contacting the originating server 210-212. FIG. 3 illustrates a domain name server (DNS) redirection mechanism that illustrates how a client 205 is connected to a front-end 201. The DNS systems is defined in a variety of Internet Engineering Task Force (IETF) documents such as RFC0883, RFC 1034 and RFC 1035 which are incorporated by reference herein. In a typical environment, a client 205 executes a browser 301, TCP/IP stack 303, and a resolver 305. For reasons of performance and packaging, browser 301, TCP/IP stack 303 and resolver 305 are often grouped together as routines within a single software product.

Browser 301 functions as a graphical user interface to implement user input/output (I/O) through monitor 311 and associated keyboard, mouse, or other user input device (not shown). Browser 301 is usually used as an interface for web-based applications, but may also be used as an interface for other applications such as email and network

news, as well as special-purpose applications such as database access, telephony, and the like. Alternatively, a special-purpose user interface may be substituted for the more general-purpose browser 301 to handle a particular application.

TCP/IP stack 303 communicates with browser 301 to convert data between formats suitable for browser 301 and IP format suitable for Internet traffic. TCP/IP stack also implements a TCP protocol that manages transmission of packets between client 205 and an Internet service provider (ISP) or equivalent access point. IP protocol requires that each data packet include, among other things, an IP address identifying a destination node. In current implementations the IP address comprises a 32-bit value that identifies a particular Internet node. Non-IP networks have similar node addressing mechanisms. To provide a more user-friendly addressing system, the Internet implements a system of domain name servers that map alpha-numeric domain names to specific IP addresses. This system enables a name space that provides a more consistent reference between nodes on the Internet and avoids the need for users to know network identifiers, addresses, routes and similar information in order to make a connection.

The domain name service is implemented as a distributed database managed by domain name servers (DNSs) 307 such as DNS_A, DNS_B and DNS_C shown in FIG. 3. Each DNS relies on <domain name:IP> address mapping data stored in master files scattered through the hosts that use the domain system. These master files are updated by local system administrators. Master files typically comprise text files that are read by a local name server, and hence

become available through the name servers 307 to users of the domain system.

The user programs (e.g., clients 205) access name servers through standard programs such as resolver 305. Resolver 305 includes an address of a DNS 307 that serves as a primary name server. When presented with a reference to a domain name for a data server 210-212, resolver 305 sends a request to the primary DNS (e.g., DNS_A in FIG. 3). The primary DNS 307 returns either the IP address mapped to that domain name, a reference to another DNS 307 which has the mapping information (e.g., DNS_B in FIG. 3), or a partial IP address together with a reference to another DNS that has more IP address information. Any number of DNS-to-DNS references may be required to completely determine the IP address mapping.

In this manner, the resolver 305 becomes aware of the IP address mapping which is supplied to TCP/IP component 303. Client 205 may cache the IP address mapping for future use. TCP/IP component 303 uses the mapping to supply the correct IP address in packets directed to a particular domain name so that reference to the DNS system need only occur once.

In accordance with the present invention, at least one DNS server 307 is owned and controlled by system components of the present invention. When a user accesses a network resource (e.g., a database), browser 301 contacts the public DNS system to resolve the requested domain name into its related IP address in a conventional manner. In a first embodiment, the public DNS performs a conventional DNS resolution directing the browser to an originating server 210-212 and server 210-212 performs a redirection of the browser to the system owned DNS server (i.e., DNC_C in FIG. 3). In a second embodiment,

domain:address mappings within the DNS system are modified such that resolution of the of the originating server's domain automatically return the address of the system-owned DNS server (DNS_C). Once a browser is redirected to the system-owned DNS server, it begins a process of further redirecting the browser 301 to the best available front-end 201.

Unlike a conventional DNS server, however, the system-owned DNS_C in FIG. 3 receives domain:address mapping information from a redirector component 309. Redirector 309 is in communication with front-end manager 207 and back-end manager 209 to obtain information on current front-end and back-end assignments to a particular server 210-212. A conventional DNS is intended to be updated infrequently by reference to its associated master file. In contrast, the master file associated with DNS_C is dynamically updated by redirector 309 to reflect current assignment of front-end 201 and back-end 203. In operation, a reference to data server 210-212 may result in an IP address returned from DNS_C that points to any selected front-end 201 that is currently assigned to data server 210-212. Likewise, data server 210-212 can identify a currently assigned back-end 203 by direct or indirect reference to DNS_C.

Front-end 201 typically receives information directly from front-end manager 207 about the address of currently assigned back-ends 203. Similarly, back-end 203 is aware of the address of a front-end 201 associated with each data packet. Hence, reference to the domain system is not required to map a front-end 201 to its appropriate back-end 203.

FIG. 4A illustrates a first embodiment in which a single intermediary computer 206 is used, whereas FIG. 4B

and FIG. 5 illustrate a second embodiment where both front-end 201 and back-end 203 are used to implement the intermediary server. Because of their similarities, FIG. 4A and FIG. 4B are described together with their differences noted.

Primary functions of the intermediary server 206 (FIG. 4A) and front-end 201 (FIG. 4B) and in include serving as a proxy for data server 210-212 from the perspective of client 205, prioritizing amongst multiple queries, and applying the queries to data servers in an order based upon the prioritization. It is contemplated that the various functions described in reference to the specific examples may be implemented using a variety of data structures and programs operating at any location in a distributed network. For example, a front-end 201 or intermediary server 206 may be operated on a network appliance 107 or server within a particular network 102, 103, or 104 shown in FIG. 1. The present invention is readily adapted to any application where multiple clients are coupling to a centralized resource. Moreover, other transport protocols may be used, including proprietary transport protocols.

TCP component 401 includes devices for implementing physical connection layer and Internet protocol (IP) layer functionality. Current IP standards are described in IETF documents RFC0791, RFC0950, RFC0919, RFC0922, RFC792, RFC1112 that are incorporated by reference herein. For ease of description and understanding, these mechanisms are not described in great detail herein. Where protocols other than TCP/IP are used to couple to a client 205, TCP component 401 is replaced or augmented with an appropriate network protocol process.

TCP component 401 communicates TCP packets with one or more clients 205. Received packets are coupled to parser 402 where the Internet protocol (or equivalent) information is extracted. TCP is described in IETF RFC0793 which is incorporated herein by reference. Each TCP packet includes header information that indicates addressing and control variables, and a payload portion that holds the user-level data being transported by the TCP packet. The user-level data in the payload portion typically comprises a user-level network protocol datagram.

Parser 402 analyzes the payload portion of the TCP packet. In the examples herein, HTTP is employed as the user-level protocol because of its widespread use and the advantage that currently available browser software is able to readily use the HTTP protocol. In this case, parser 402 comprises an HTTP parser. More generally, parser 402 can be implemented as any parser-type logic implemented in hardware or software for interpreting the contents of the payload portion. Parser 402 may implement file transfer protocol (FTP), mail protocols such as simple mail transport protocol (SMTP) and the like. Any user-level protocol, including proprietary protocols, may be implemented within the present invention using appropriate modification of parser 402.

To improve performance, front-end 201 optionally includes a caching mechanism 403. Cache 403 may be implemented as a passive cache that stores frequently and/or recently accessed database content or as an active cache that stores database content that is anticipated to be accessed. In non-web applications, cache 403 may be used to store any form of data representing database contents, files, program code, and other information.

Upon receipt of a TCP packet, HTTP parser 402 determines if the packet is making a request for data within cache 403. If the request can be satisfied from cache 403 the data is supplied directly without reference to data server 210-212 (i.e., a cache hit). Cache 403 implements any of a range of management functions for maintaining fresh content. For example, cache 403 may invalidate portions of the cached content after an expiration period specified with the cached data or by data sever 210-212. Also, cache 403 may proactively update the cache contents even before a request is received for particularly important or frequently used data from data server 210-212. Cache 403 evicts information using any desired algorithm such as least recently used, least frequently used, first in/first out, or random eviction. When the requested data is not within cache 403, a request is processed to data server 210-212, and the returned data may be stored in cache 403.

Several types of packets will cause parser 404 to forward a request towards data server 210-212. For example, a request for data that is not within cache 403 (or if optional cache 403 is not implemented) will require a reference to data server 210-212. Some packets may comprise data that may be supplied to data server 210-212 (e.g., customer credit information, form data and the like). In these instances, HTTP parser 402 couples to data blender 404.

In the embodiment of FIG. 4A, the intermediary server 206 may be located topologically near the client 205 or data server 210-212 --either alternative provides some advantage and the choice of location is made to meet the needs of a particular application. Query language processor 408 receives a parsed request and formulates it

into a proper (i.e., syntactically correct) database query.

The formulated query is passed to transport component 409 for communication to data server 210-212 over channel 411. Channel 411 is compatible with an interface to data server 210-212 which may include a TCP/IP interface as well as Ethernet, Fibre channel, or other available physical and transport layer interfaces. In a particular example, transport component 409 is implemented using extensible data server interface such as the Java database components (JDBC) that enable plug-in extensions to support particular database formats.

Data server returns responses to transport component 409 and supplies them to data filter 406 and/or HTTP format component 407. Data filter component may filter and/or constrain database contents returned in the response. Data filter implements these functions typically implemented in a DBMS. Data filter component 406 is optionally used to implement data decompression where appropriate, decryption, and handle caching when the returning data is of a cacheable type. HTTP format component 407 formats the response into a format suitable for use by client 205, which in the particular examples herein comprises a web page transported as an HTTP packet.

Where two intermediary computers are used as in the example of FIG. 4B and FIG. 5, front-end 201 is responsible for translating transmission control protocol (TCP) packets from client 205 into transport morphing protocol™ (TMP™) packets used in the system in accordance with the present invention. Transport morphing protocol and TMP are trademarks or registered trademarks of Circadence Corporation in the United States and other countries. Query formation may take place in back-end

203. Query language processing may require knowledge of the database structure and schema of the target data server 210-212. This knowledge will take the form of a mapping table, for example, that maps field identifications within a data server 210 to fields present in parsed request packets. Such information may be more readily applied in a back-end 203. Conversely, HTTP formatting component 407 is more usefully is preferably implemented in front-end 201 where knowledge of a particular client 205 may be more readily available.

Optionally, front-end 201, back end 203, and intermediary computer 206 implement security processes, compression processes, encryption processes and the like to condition the received data for improved transport performance and/or provide additional functionality. These processes may be implemented within any of the functional components (e.g., data blender 404) or implemented as separate functional components within front-end 201. Also, parser 402 may identify priority information transmitted with a request. The prioritization value may be provided by the owners of data server 210-212, for example, and may be dynamically altered, statically set, or updated from time to time to meet the needs of a particular application. Moreover, priority values may be computed to indicate aggregate priority over time, and/or combine priority values from different sources to compute an effective priority for each database request.

In the embodiment of FIG. 4B and FIG. 5, blender 404 slices and/or coalesces the data portions of the received packets into a more desirable "TMP units" that are sized for transport through the TMP mechanism 202. The data portion of TCP packets may range in size depending on

client 205 and any intervening links coupling client 205 to TCP component 401. Moreover, where compression is applied, the compressed data will vary in size depending on the compressibility of the data. Data blender 404 receives information from front-end manager 207 that enables selection of a preferable TMP packet size. Alternatively, a fixed TMP packet size can be set that yields desirable performance across TMP mechanism 202. Data blender 404 also marks the TMP units so that they can be re-assembled at the receiving end.

Data blender 404 may also serve as a buffer for storing packets from all appliances 177 that are associated with front-end 201. In accordance with the present invention, data blender 404 may associate a prioritization value with each packet.

TMP mechanisms 405 and 505 implement the TMP protocol in accordance with the present invention. TMP is a TCP-like protocol adapted to improve performance for multiple channels operating over a single connection. Front-end TMP mechanism 405 and corresponding back-end TMP mechanism 505 shown in FIG. 5 are computer processes that implement the end points of TMP link 202. The TMP mechanism in accordance with the present invention creates and maintains a stable connection between two processes for high-speed, reliable, adaptable communication.

Another feature of TMP is its ability to channel numerous TCP connections through a single TMP pipe 202. The environment in which TMP resides allows multiple TCP connections to occur at one end of the system. These TCP connections are then combined into a single TMP connection. The TMP connection is then broken down at the other end of the TMP pipe 202 in order to traffic the TCP connections to their appropriate destinations. TMP

includes mechanisms to ensure that each TMP connection gets enough of the available bandwidth to accommodate the multiple TCP connections that it is carrying.

Another advantage of TMP as compared to traditional protocols is the amount of information about the quality of the connection that a TMP connection conveys from one end to the other of a TMP pipe 202. As often happens in a network environment, each end has a great deal of information about the characteristics of the connection in one direction, but not the other. By knowing about the connection as a whole, TMP can better take advantage of the available bandwidth.

FIG. 5 illustrates principle functional components of an exemplary back-end 203 in greater detail. Primary functions of the back-end 203 include serving as a proxy for client 205 from the perspective of data server 210, translating transmission control protocol (TCP) packets from data server 210 into TMP packets as well as translating TMP packets from FE201 into the one or more corresponding TCP packets generated by clients 205.

TMP unit 505 receives TMP packets from TMP pipe 202 and passes them to HTTP reassemble unit 507 where they are reassembled into the corresponding TCP packets. Data filter 506 may implement other functionality such as decompression, decryption, and the like to meet the needs of a particular application. The reassembled data is forwarded to TCP component 501 for communication with data server 210-212.

TCP data generated by the data server process are transmitted to TCP component 501 and forwarded to data filter 502. Data filter 502 operates in a manner analogous to data filter 406 shown in FIG. 4A and FIG. 4B.

Data blender 504 operates in a manner akin to data blender 404 shown in FIG. 4B to buffer and prioritize packets in a manner that is efficient for TMP transfer. Priority information is received by, for example, back-end manager 209 based upon criteria established by the web site owner. TMP data is streamed into TMP unit 505 for communication on TMP pipe 202.

The present invention also involves a number of alternative modes of functioning using the components shown in FIG. 2A and FIG. 2B. In a first alternative, the front-ends 201 and/or intermediate computers 206 receive requests and generate database-specific queries to cause the database to retrieve the desired data. The data is supplied directly to the front-ends 201 and/or intermediate computers 206 which execute web server processes to generate markup language documents (e.g., HTML documents). The markup language documents are sent to requesting client applications.

In a second alternative, the front-end 201 or intermediary 206 receives requests, and generates a remote procedure call to a back-end 203 or data server 210-212 having access to a database. In this alternative, the back-end server 203 or data server 210-212 formulates and executes a query against the database and returns the data to the front-end computer 201 or intermediary computer 206. This alternative avoids a requirement for the front-end 201 and/or intermediary 206 to have special knowledge of the database structure needed to formulate valid queries against a particular database. However, this second alternative also enables the processing load required to generate pages to be separated from the database operations.

In a third alternative, the front-end 201 and/or intermediary 206 receives a request and notifies an appropriate data server 210-212. The data server returns the specified data along with issuing a remote procedure call to the front-end 201 or intermediary 206. The remote procedure call causes the front-end 201 or intermediary 206 to generate the web page using the returned data. This implementation has an advantage that the data server 210-212 remains in control of the page generation process, but off-loads some of the processing load required to generate a page to the front-end computer 201 or intermediary computer 206.

In a fourth alternative, a front-end computer receives a request and notifies an appropriate data server 210-212. The data server responds to the notification by sending a remote procedure call to the front-end computer 201 or intermediary computer 206. The remote procedure call is accompanied by parameters that enable the front-end computer 201 or intermediary computer 206 to execute the remote procedure to generate a properly formatted database query that is applied to a database directly. Like the second alternative, this avoids need for the front-end 201 or the intermediary computer 26 to have specialized knowledge of the database implementation as that knowledge is conveyed as needed by the remote procedure call.

In yet a fifth alternative, the data server 210-212 provides a profile of the target database in the form of parameters to the front-end server 201 or the intermediary computer 26. The profile is used by the front-end server 201 or the intermediary computer 26 to generate a properly formed database query. In either the third or fourth

alternative, it is not necessary that the database be logically coupled or co-located with the data server.

One application of the methods and systems of the present invention is to provide alternative content in response to a user request when the desired content is unavailable. A particular brand of alternative content provision is Soft Landing™ service. Soft Landing is a trademark of Circadence Corporation in the United States and other countries. In accordance with the present invention, in cases where an originating web server cannot respond to a request, the present invention enables a front-end or intermediary server to make database requests to obtain content from alternative sources and construct a response on behalf of the unavailable originating web server.

The alternative sources may provide identical content or varying content. For example, when the originating web server is unavailable because it lacks resources to generate a web page, the present invention enables the front-end or intermediary server to access the same data as the originating web server and generate the web page without using the originating web server's resources. Alternatively, the front-end or intermediary server may access a mirror copy of the originating web server's data, or may access an entirely separate database to obtain truly alternative content.

Although the invention has been described and illustrated with a certain degree of particularity, it is understood that the present disclosure has been made only by way of example, and that numerous changes in the combination and arrangement of parts can be resorted to by those skilled in the art without departing from the spirit and scope of the invention, as hereinafter claimed. For

example, while devices supporting HTTP data traffic are used in the examples, the HTTP devices may be replaced or augmented to support other public and proprietary protocols including FTP, NNTP, SMTP, SQL and the like. In such implementations the front-end 201 and/or back end 203 are modified to implement the desired protocol. Moreover, front-end 201 and back-end 203 may support different protocols such that the front-end 201 supports, for example, HTTP traffic with a client and the back-end supports a DBMS protocol such as SQL. Such implementations not only provide the advantages of the present invention, but also enable a client to access a rich set of network resources with minimal client software.